M11-L2 Problem 1

In this problem you will implement the elbow method using three different sklearn clustering algorithms: (`KMeans` , `SpectralClustering` , `GaussianMixture`). You will use the algorithms to find the number of natural clusters for two different datasets, one "blob" shaped dataset, and one concetric circle dataset.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200

from sklearn.datasets import make_blobs, make_circles
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.mixture import GaussianMixture

import warnings
warnings.filterwarnings('ignore')

def plot_loss(loss, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    ax.plot(np.arange(2, len(loss)+2), loss, 'k-o')
    ax.set_xlabel('Number of Clusters')
    ax.set_ylabel('Loss')
    if title:
        ax.set_title(title)
    return ax

def plot_pred(x, labels, ax = None, title = None):
    if ax is None:
        ax = plt.gca()
    n_clust = len(np.unique(labels))
    for i in range(n_clust):
        ax.scatter(x[labels == i,0], x[labels == i,1], alpha = 0.5)
    ax.set_title(title)
    return ax

def compute_loss(x, labels):
    # Initialize loss
    loss = 0
    # Number of clusters
    n_clust = len(np.unique(labels))
```

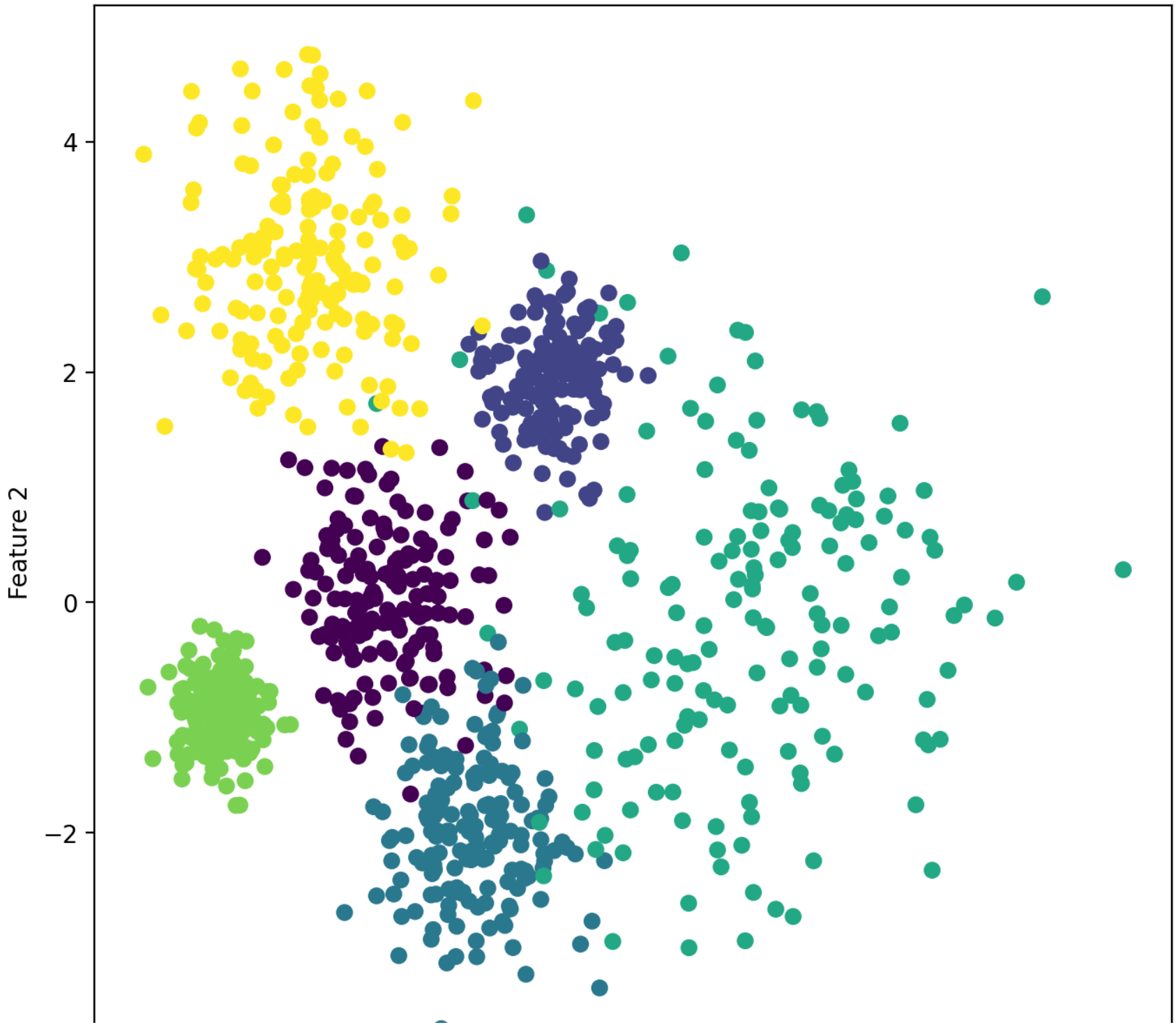
```
    # Loop through the clusters
    for i in range(n_clust):
        # Compute the center of a given label
        center = np.mean(x[labels == i, :], axis = 0)
        # Compute the sum of squared distances between each point and its corresponding cluster center
        loss += np.sum(np.linalg.norm(x[labels == i, :] - center, axis = 1)**2)
    return loss
```

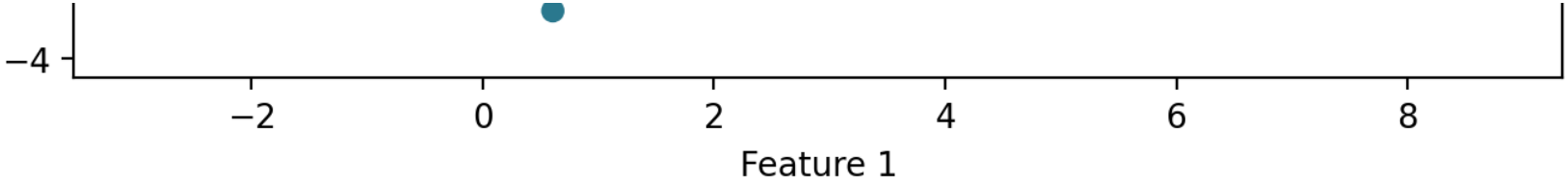
Blob dataset

Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where `y` contains the label of each corresponding point in `x`.

In [2]:
```
## DO NOT MODIFY
x, y = make_blobs(n_samples = 1000, n_features = 2, centers = [[0,0],[2,2],[1,-2],[4,0],[-2,-1],[-1,3]], cluster_std =
```

In [3]:
```
## YOUR CODE GOES HERE

plt.figure(figsize=(8,8))
scatter = plt.scatter(x[:, 0], x[:, 1], c=y)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
```

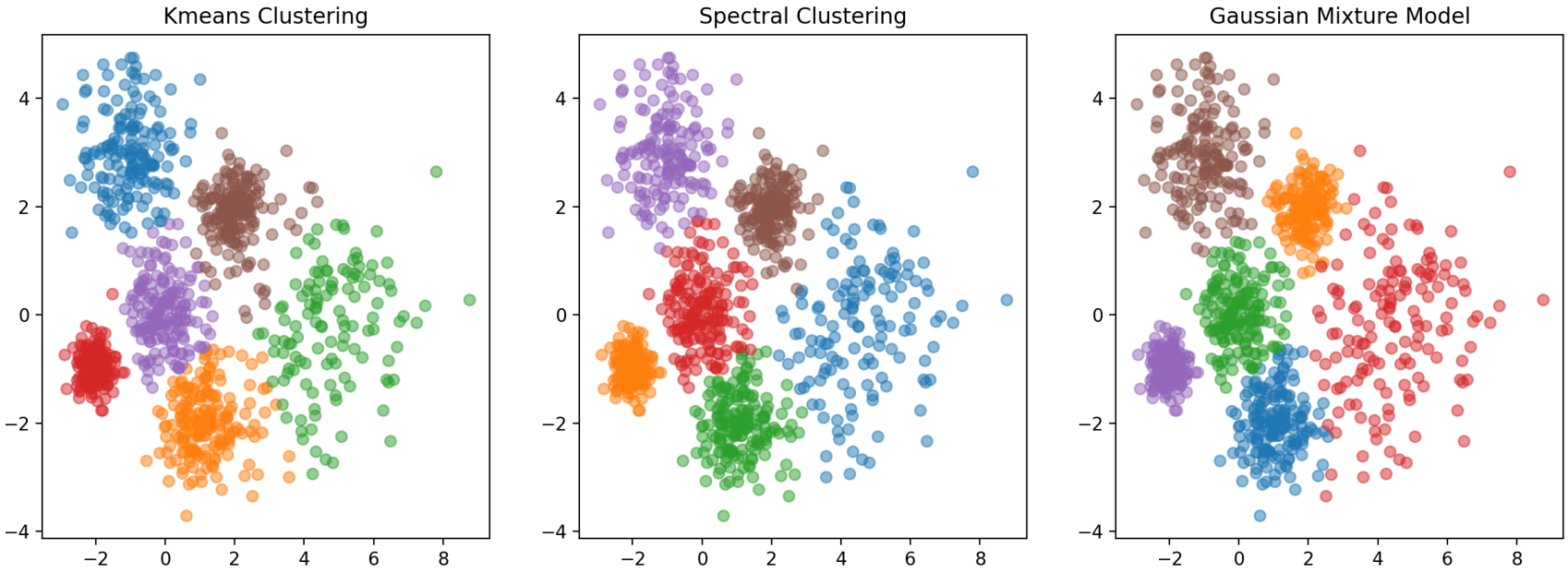

Feature 2
4
2
0
−2



Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the "blob" data with 6 clusters, and modify the parameters until you get satisfactory results. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

In [4]:
```
## YOUR CODE GOES HERE

fig, ax = plt.subplots(1,3,figsize = (15,5))

myKmeans = KMeans(n_clusters = 6, n_init = 10)
labels_1 = myKmeans.fit_predict(x)
plot_pred(x,labels_1,ax[0],title = "Kmeans Clustering")

mySpectral = SpectralClustering(n_clusters = 6, n_init = 10, affinity = 'nearest_neighbors')
labels_2 = mySpectral.fit_predict(x)
plot_pred(x,labels_2,ax[1],title = "Spectral Clustering")

myGaussian = GaussianMixture(n_components = 6, n_init = 10)
labels_3 = myGaussian.fit_predict(x)
plot_pred(x,labels_3,ax[2],title = "Gaussian Mixture Model")
```

Out[4]:
```
<Axes: title={'center': 'Gaussian Mixture Model'}>
```

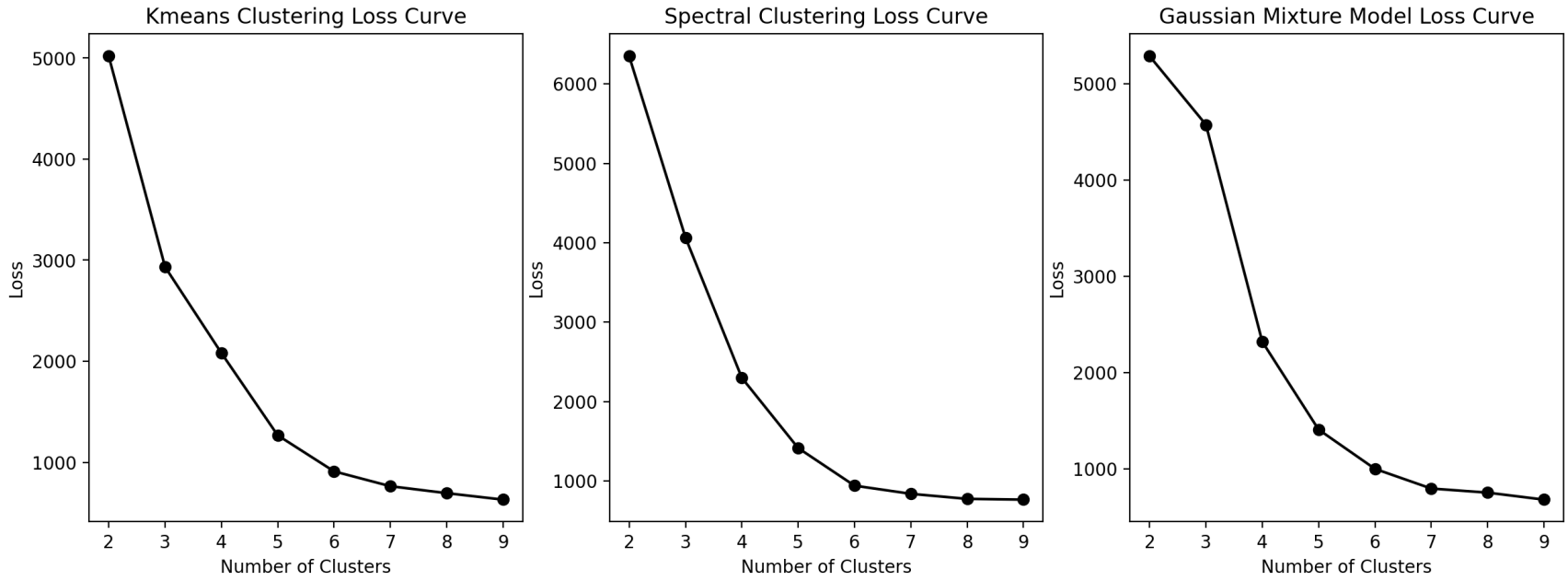


Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_pred(x, labels, ax, title)` function.

In [5]:
```
## YOUR CODE GOES HERE

n_clust = [2,3,4,5,6,7,8,9]


kmeans_loss = []
spectral_loss = []
gaussian_loss = []

for n in n_clust:
    myKmeans = KMeans(n_clusters = n, n_init = 10)
    labels_kmeans = myKmeans.fit_predict(x)
    loss_k = compute_loss(x,labels_kmeans)
    kmeans_loss.append(loss_k)

    mySpectral = SpectralClustering(n_clusters = n,n_init = 10, affinity = 'nearest_neighbors')
    labels_spectral = mySpectral.fit_predict(x)
    loss_s = compute_loss(x,labels_spectral)
```

```
    spectral_loss.append(loss_s)

    myGaussian = GaussianMixture(n_components = n,n_init = 10)
    labels_gaussian = myGaussian.fit_predict(x)
    loss_g = compute_loss(x,labels_gaussian)
    gaussian_loss.append(loss_g)

fig, ax = plt.subplots(1,3,figsize = (15,5))
plot_loss(kmeans_loss,ax[0],"Kmeans Clustering Loss Curve")
plot_loss(spectral_loss,ax[1],"Spectral Clustering Loss Curve")
plot_loss(gaussian_loss,ax[2],"Gaussian Mixture Model Loss Curve")
```

Out[5]:
```
<Axes: title={'center': 'Gaussian Mixture Model Loss Curve'}, xlabel='Number of Clusters', ylabel='Loss'>
```



Concentric circles dataset

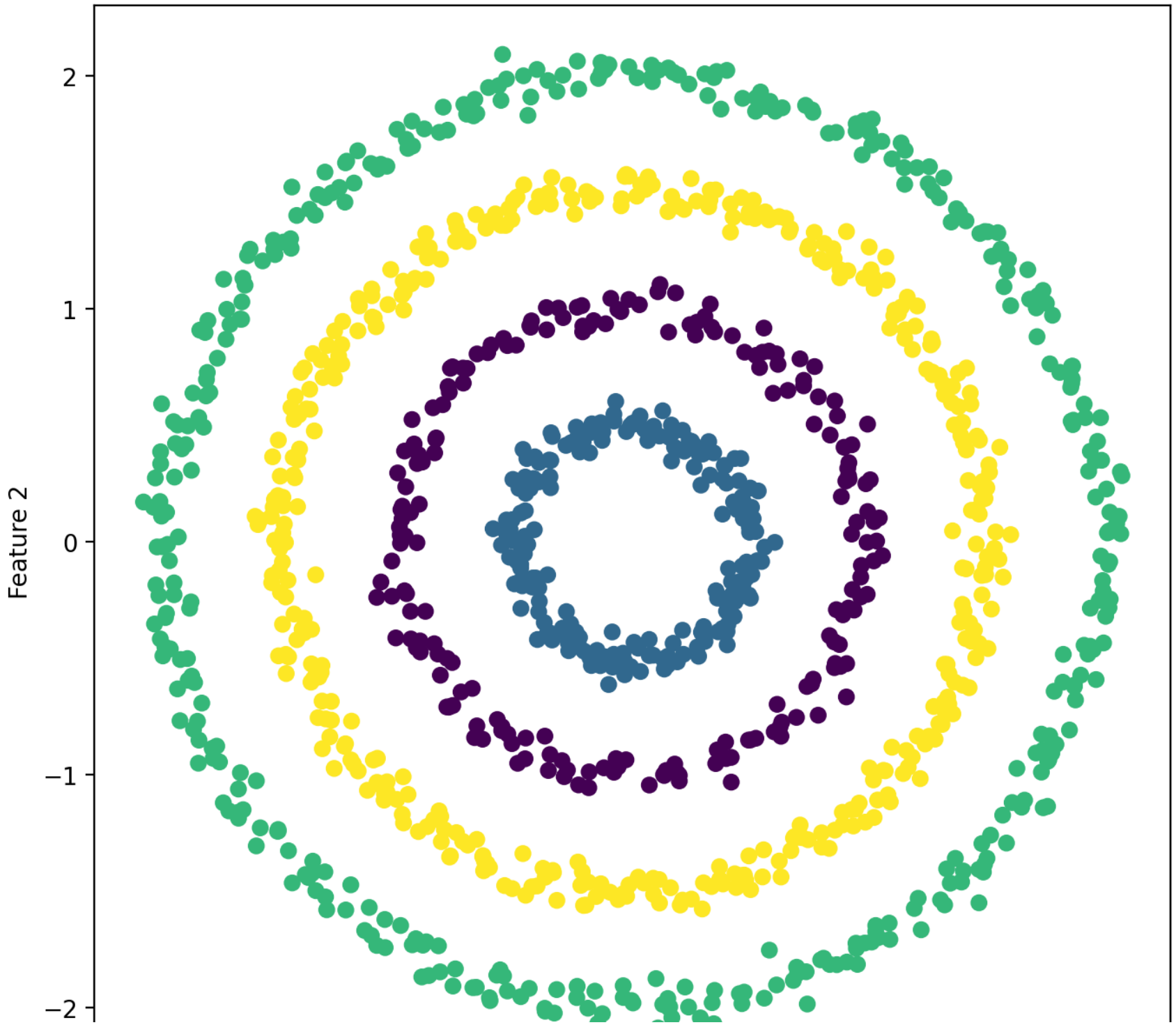
Visualize the "blob" dataset generated below, using a unique color for each cluster of points, where `y` contains the label of each corresponding point in `x`.

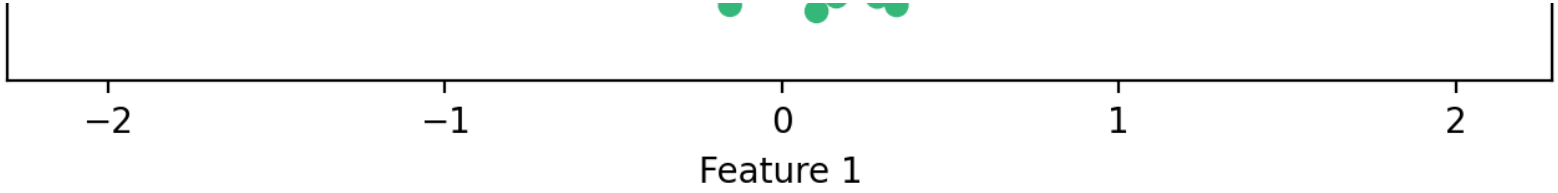
In [6]:
```
## DO NOT MODIFY
x1, y1 = make_circles(n_samples = 400, noise = 0.05, factor = 0.5, random_state = 0)
x2, y2 = make_circles(n_samples = 800, noise = 0.025, factor = 0.75, random_state = 1)
```

```
x = np.vstack([x1, x2*2])
y = np.hstack([y1, y2+2])
```

In [7]:

```
## YOUR CODE GOES HERE

plt.figure(figsize=(8,8))
scatter = plt.scatter(x[:, 0], x[:, 1], c=y)

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')

plt.show()
```

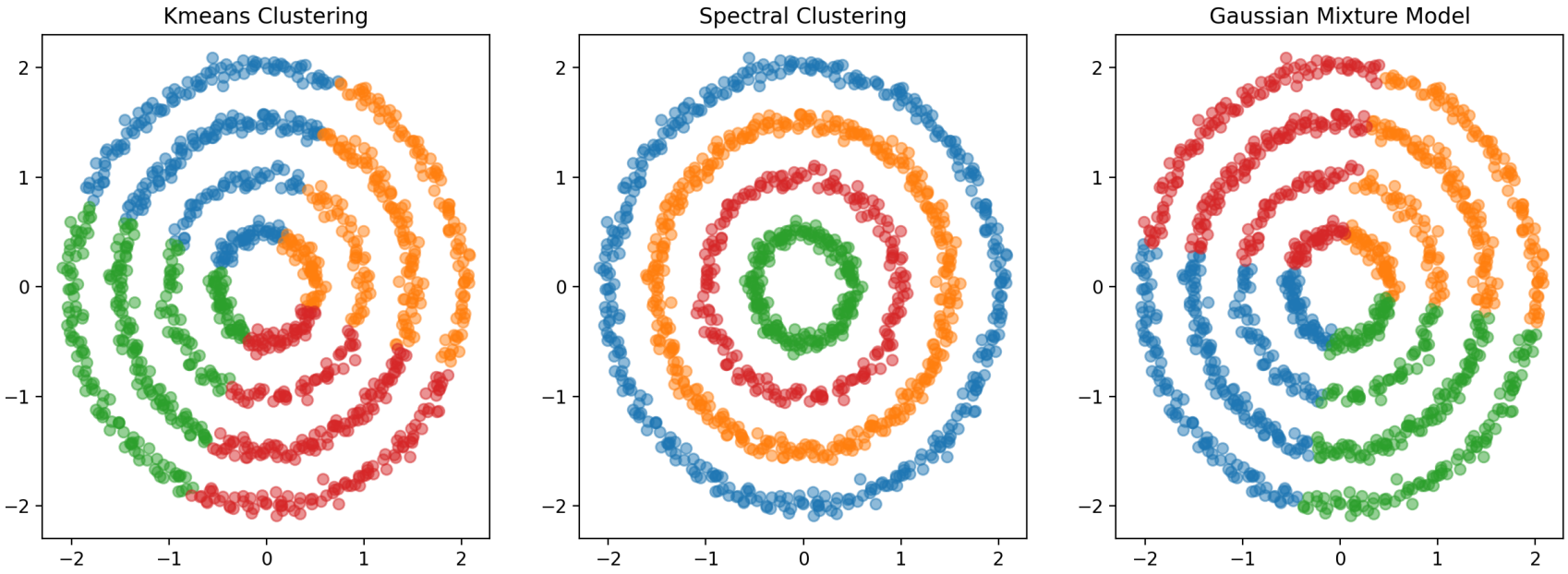

Feature 2
2
1
0
−1
−2



Use the `sklearn` KMeans, Spectral Clustering, and Gaussian Mixture Model functions to cluster the concentric circle data with 4 clusters, and attempt to modify the parameters until you get satisfactory results. Note: you should get good clustering results with Spectral Clustering, but the KMeans and GMM models will struggle to cluster this dataset well. Plot the results of your three models side-by-side using `plt.subplots` and the provided `plot_pred(x, labels, ax, title)` function.

In [11]:
```
## YOUR CODE GOES HERE

fig, ax = plt.subplots(1,3,figsize = (15,5))

myKmeans = KMeans(n_clusters = 4)
labels_1 = myKmeans.fit_predict(x)
plot_pred(x,labels_1,ax[0],title = "Kmeans Clustering")

mySpectral = SpectralClustering(n_clusters = 4,  affinity = 'nearest_neighbors')
labels_2 = mySpectral.fit_predict(x)
plot_pred(x,labels_2,ax[1],title = "Spectral Clustering")

myGaussian = GaussianMixture(n_components = 4)
labels_3 = myGaussian.fit_predict(x)
plot_pred(x,labels_3,ax[2],title = "Gaussian Mixture Model")
```

Out[11]:
```
<Axes: title={'center': 'Gaussian Mixture Model'}>
```

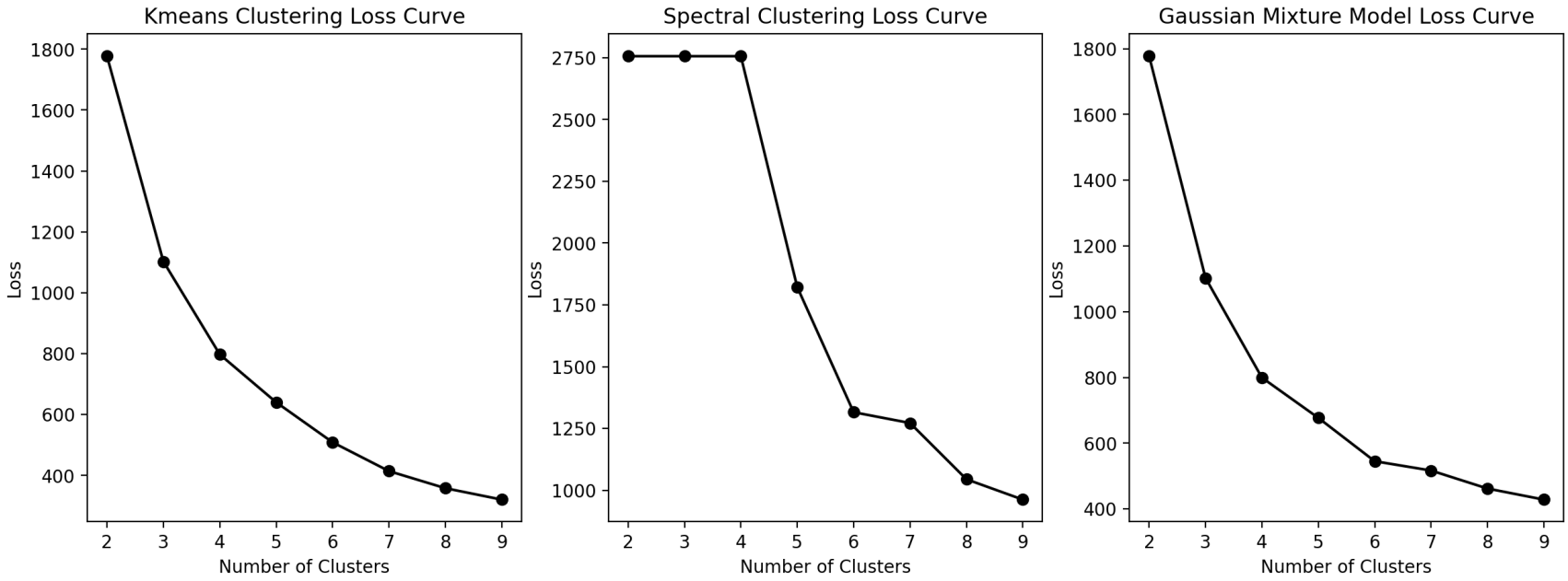


Using the parameters you found for the three models above, run each of the clustering algorithms for `n_clust = [2,3,4,5,6,7,8,9]` and compute the sum of squared distances loss for each case using the provided `compute_loss(x, labels)` function, where labels is the cluster assigned to each point by the algorithm. Plot loss versus number of cluster for each your three models in side-by-side subplots using the provided `plot_pred(x, labels, ax, title)` function.

In [12]:
```
## YOUR CODE GOES HERE

n_clust = [2,3,4,5,6,7,8,9]


kmeans_loss = []
spectral_loss = []
gaussian_loss = []

for n in n_clust:
    myKmeans = KMeans(n_clusters = n, n_init = 10)
    labels_kmeans = myKmeans.fit_predict(x)
    loss_k = compute_loss(x,labels_kmeans)
    kmeans_loss.append(loss_k)

    mySpectral = SpectralClustering(n_clusters = n, n_init = 10, affinity = 'nearest_neighbors')
    labels_spectral = mySpectral.fit_predict(x)
    loss_s = compute_loss(x,labels_spectral)
```

```
    spectral_loss.append(loss_s)

    myGaussian = GaussianMixture(n_components = n, n_init = 10)
    labels_gaussian = myGaussian.fit_predict(x)
    loss_g = compute_loss(x,labels_gaussian)
    gaussian_loss.append(loss_g)

fig, ax = plt.subplots(1,3,figsize = (15,5))
plot_loss(kmeans_loss,ax[0],"Kmeans Clustering Loss Curve")
plot_loss(spectral_loss,ax[1],"Spectral Clustering Loss Curve")
plot_loss(gaussian_loss,ax[2],"Gaussian Mixture Model Loss Curve")
```

Out[12]: <Axes: title={'center': 'Gaussian Mixture Model Loss Curve'}, xlabel='Number of Clusters', ylabel='Loss'>



Discussion

1. Discuss the performance of the clustering algorithms on the "blob" dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods? Does the elbow method work better for some algorithms versus others?

All three algorithms worked well with the blob dataset. The elbow method gave us very ambiguous results. In Kmeans and spectral clustering the number of clusters as seen from the loss curve can be 5 or 6 which is not clear as we don't have enough information but for the GMM it is pretty clear that the number of clusters is 6 as the loss reduces slowly after 6.

1. Discuss the performance of the clustering algorithms on the concentric circles dataset. Using the elbow method, were you able to identify the number of natural clusters in the dataset for each of the methods?

For concentric circles dataset the spectral clustering performs better than others. The Kmeans and GMM clustered in a wrong way. For Kmeans the elbow method tells that the number of natural clusters is between 4 and 7. For Spectral clustering the elbow method tells that the number of natural clusters is 5 or 6 and for GMM the elbow method tells that the number of natural clustering is around 4 which tells that the elbow method worked well for GMM than others

1. Does the sum of squared distances work well as a loss function for each of the three clustering algorithms we implemented? Does the sum of squared distance fail on certain types of clusters?

The sum of squared distances worked well for the blob dataset for all the three algorithms because it is a natural clustering criterion. But for concentric circles dataset, the loss curves are not good which suggests that the method didn't work well. This can be because the clustering needs to happen by latching onto chains. Kmeans and Gaussian worked better in concentric circles dataset as compared to spectral clustering.

In [ ]: